The Research Software Engineer

The Research Software Engineer

Rob Baxter[1], Neil Chue Hong[1]; Dirk Gorissen[2]; James Hetherington[3]; Ilian Todorov[4]

[1] Software Sustainability Institute & University of Edinburgh; [2] University of Southampton; [3] UCL; [4] STFC Daresbury Laboratory

Background

Research is increasingly digital. Twenty-first century research has been characterised by the rise of digital methods, the third and fourth paradigms of science – computational simulation and data-intensive research. In their turn, these new approaches are both built on a common foundation – computer software.

Yet despite this increasing reliance on software in research, professional practices for developing research software in academia lag far behind those in the commercial sector. Computational research tools are often fragile, generally not sustainable or usable beyond the lifetime of a given project, and frequently unsuitable for scrutiny. Those trained solely within academia often employ ad-hoc or casual development techniques. Institutions miss out on opportunities to increase the impact of their research by producing robust software deliverables that could be used and cited by their peers.

Computational work must reflect the committed attitude of experimentalists towards caring about precise, professional, repeatable, meticulous work – no-one with the same casual attitude to experimental instrumentation as many researchers have to code would be allowed anywhere near a lab. This is striking considering how often research results now depend on software.

Software engineering professionals are trained in best practices, and in the best commercial institutions follow a disciplined approach to the design, construction, testing and maintenance of software systems. Attempts to leverage these skills within academia by employing contract programmers typically fail, due to otherwise talented programmers lacking sufficient research experience and a necessary appreciation of the significant cultural differences between business and academia. Software engineers that do have research experience and good knowledge of the underlying domain are, however, in very short supply, due to the lack of appropriate institutional homes and career progression paths for their work.

This paper provides a synthesis of discussions that took place during and after the 2012 Collaborations Workshop organized by the Software Sustainability Institute in Oxford, UK.

The Research Software Engineer

Research institutions need individuals with a new professional designation – the *research software engineer*. These individuals combine a professional attitude to the exercise of software engineering with a deep understanding of research topics. They lead the design and construction of increasingly complex research software systems, and play an important part in the co-design of research requirements, understanding and addressing software engineering questions that arise in research planning.

To successfully and usefully apply software development skills in research, software engineers must be able to understand the domain within which they work. Understanding research literature is a major part of the life of the research software engineer. Scientific software developers primarily create software designs and code, but remain an integral part of the research community, with authorship on research papers. Their work cannot be simplified to that of a technician or a contract programmer as these engineers have roles closer to those of researchers with a complementary skill set.

Research software engineers test their hypotheses and solutions for correctness by subjecting them to well-defined test cases. They keep abreast of the cutting edge scientific methodologies in order to develop and prove them worthwhile. They develop pedagogic skills to teach and train colleagues. They can write papers and grant proposals, as well as technical reports and manuals. They keep abreast of relevant IT trends – numerical algorithms, computer languages, analysis tools and hardware.

Although all research software engineers must combine deep background in science with strong software development skills, there exists nevertheless a spectrum from those whose interests focus on software development as a means to scientific ends, to those for whom the practice of scientific software development is their primary focus. At the two ends of this spectrum, we see:

- **the "Researcher-Developer"** wants to be judged on their scientific output, is a researcher at heart, however develops a lot of code due to the nature of their research. They would like recognition comparable to a research paper for a software implementation taken up by others.
- **the "Research Software Engineer"** comes from a research background but is also a skilled software developer, and relishes challenge of not just developing code to solve a problem but doing it well. They want to be recognised for producing tools which others rely on for research.

Career Development

Research has changed enormously in the last 20 years, but career progression continues to be based on "publication performance", as measured by journal research papers. At the same time, software development has diversified, becoming a well-defined profession with many sub-fields. This profession carries complex intellectual demands, and commercial software engineers expect compensation and recognition appropriate to their status as skilled professionals. Software engineers in academia must be able to achieve comparable recognition and status to their academic and industrial colleagues.

The training required to be an effective research software engineer takes at least a decade. Individuals will typically complete a research training programme through PhD and postdoc, and also complete an equivalently-demanding training programme in software development, often involving time spent in a commercial software development company.

At present there is no progression path within academic institutions for such individuals. An inability to progress beyond PDRA makes talented developers leave. This is a major loss to research institutions of skills investment made to gain expertise in both research and software development.

This absence of a career path is due to the lack of recognition and appreciation of the output of software development work in the academic system of assessment, which measures success only through published peer-reviewed publications. The Science Code Manifesto [http://sciencecodemanifesto.org/], whose founding signatories include Peter Norvig, Joseph Jackson, and Victoria Stodden, talks about this explicitly:

> "*Software is an essential research product, and the effort to produce, maintain, adapt, and curate code must be recognized. Software stands among other vital scientific contributions besides published papers. "*

Funding Research Software Development

Until recently, software development in academia has been viewed as an uninteresting means of achieving interesting research. Research Councils have had no funding policy for software development and sustainability, which has led to the need to disguise such development in grant proposals. Software development has been carried out in a cash-starved environment, where journeyman software developers migrate from one project to another, taking their vital software knowledge with them. Often when a research project ends, so does the work of maintaining the software.

Development focuses on meeting the minimum requirements of a specific scientific research case, rather than investing in software that is generic, robust and reusable. Without other research projects as future stakeholders, Principal Investigators can only extend the life of their software by extending the life of their project through grant renewals. All of this means that it is challenging to produce reusable software that can last beyond the scope of any one project. Some software projects stop, freeze or move overseas, leading to irreplaceable losses of the institutional memory embodied in research software developers expertise and experience.

These problems stem from the necessarily high price to pay for proper software development and support, the muddy clarity of IP and ownership of software, and the limited understanding of how to benchmark software

developers' skills. We feel that the role of the research software engineer should be explicitly recognised in calls for funding proposals, and should be explicitly valued as researcher-equivalent. It is worth pointing out the obvious fact that a PRDA salary is much less than that of a commercial software developer.

The demand for software development has risen, but the availability of scientific software developers has not, due to deficiencies in long-term planning and succession policies. Millions of pounds are spent annually on computational infrastructure in UK academia – we would argue that expenditure on individuals capable of producing quality research software could have a greater return on investment.

Signs of Change

Elements of a new best practice are beginning to emerge, elements which we hope form the beginnings of a remedy for these issues. In an effort to encourage these green shoots, we highlight a few here.

Some software development and related services have been supported by the Research Councils under programmes such as the CCPs, e-Minerals and e-Science, and initiatives like the Software Sustainability Institute. Funding has been provided by EPSRC to support applications via the CCPs; to software development via HEA (Daresbury Laboratory); to software HPC optimisation via distributed computational and software engineering (dCSE) projects serviced by NAG Ltd; and to "next generation" software codes in a number of recent calls. The BBSRC tools and resources development fund has established both the demand and an effective funding model for software development in the life sciences. This year EPSRC announced software development fellowships, which are a vital step towards allowing researchers with a significant software focus to build their careers.

On the university side, there has been some positive activity: computational science and engineering has been established as an MSc course by the EPCC and recognised as a different subject from computer science. HPC and parallel programming training courses are now widespread and are regular events for PhD students in natural sciences. A number of universities such as University College London and the University of Bristol are creating software development groups with the intent to assist researchers across departments to design, build and maintain high quality in-house software.

Recommendations

All these strategic initiatives are, of course, to be welcomed, and the authors note the encouraging progress made in recent years in addressing some of the issues we highlight. However, more needs to be done to help embed the necessary culture change across the UK academia and research institutions. We have three key recommendations:

- The REF allows for recognition of software deliverables through its system for impact measurement. In practice, however, this depends on the decisions of individual panels. *REF panels should be given clear guidance on the importance of weighing software as a "first class" research output*.
- Research software engineers are a new role in academic institutions. *Institutions and funding panels should recognise the value of this role in funding research proposals and in providing career progression and succession for such individuals*.
- All researchers must be exposed to best practices in software development. *The fundamentals of good software engineering should form part of every researcher's basic training.*

As Henry Ford said: "culture eats strategy for breakfast". We feel the time is right to accelerate the necessary cultural changes by recognising explicitly the new role of the research software engineer.